Hybrid ROI-Based Visualization of Medical Models

Lazaro Campoalegre
Trinity College Dublin
College Green
Dublin 2, Ireland
campoall@tcd.ie

Isabel Navazo
Universitat Politècnica de Catalunya
C. Jordi Girona 31
08034 Barcelona, Spain
isabel@cs.upc.edu

Pere Brunet
Universitat Politècnica de Catalunya
C. Jordi Girona 31
08034 Barcelona, Spain
pere@cs.upc.edu

ABSTRACT

There is an increasing interest on tele-medicine and tele-diagnostic solutions based on the remote inspection of volume data coming from multimodal imaging. Client-server architectures meet these functionalities. The use of mobile devices is sometimes required due to the portability and easy maintenance. However, transmission time for the volumetric information and low performance hardware properties, make quite complex the design of efficient visualization systems on these devices. In this paper, we present a hybrid approach which is based on regions of interest (ROIs) and on a transfer-function aware compression scheme. It has a good performance in terms of bandwidth requirements and storage needs in the client device, being flexible enough to represent several materials and volume structures in the ROI. Clients store a low-resolution version of the volume data and ROI-dependent high resolution segmented information. Data must be only sent whenever a new ROI is requested, but interaction in the client is autonomous - without any data transmission - while a certain ROI is inspected. A benchmark is presented to compare the the proposed scheme with three existing approaches, on two different volume data models. The results show that our hybrid approach is compact, efficient and scalable, with compression rates that decrease when the size of the volume model increases.

Keywords

Volume rendering, client-server, mobile devices, medical data, region of interest, ray-casting, volume data compression

1 INTRODUCTION

Recently, several important research areas in three-dimensional techniques for multimodal imaging have appeared. Applications include neurological imaging for brain surgery, tissue characterization, medical school teaching, plastic surgery and others. At the same time, scientists are more familiarized with three-dimensional structures reconstruction from Two-dimensional images.

The reconstruction of a volumetric model is generally achieved by using a voxel representation of datasets. According to the structure to be highlighted during the visualization, a transfer function is applied to assign color and opacity to the density value which represents the structure properties.

The handling of three-dimensional information requires efficient systems to achieve fast data transmission and interactive visualization of high quality images. Client-server applications allow these functionalities. Sometimes the use of mobile devices is necessary due to the portability and easy maintenance. However, transmission time for the volumetric information and low performance hardware properties, complicate the design of efficient visualization systems on these devices.

The main contribution of our work is a Hybrid visualization approach that inherits the advantages of some previous algorithms like the ones presented in [1] and [2], while keeping a good performance in terms of bandwidth requirements and storage needs in client devices. The scheme is flexible enough to represent several materials and volume structures in the Region of Interest (ROI) at high resolution and very limited information transmission cost.

2 PREVIOUS WORK

Client-server architectures have grown in popularity. Mobile devices as well as desktop computers can both function as clients requesting and receiving information over the network. Many authors have published research results in the remote volume visualization area. However there is still scarce specific bibliography for volume visualization in mobile devices. The majority of the proposals use known algorithms like Ray-Casting, 2D Textures, and isosurface modeling to render volume data. In order to compensate limitations in low performance devices or to reduce costs, the number of client-server schemes have been proposed.

In some client-server approaches the dataset is compressed on the server side and sent to the client where

the transfer function is applied after decompression and before the rendering of the recovered data. Moser and Weiskopf [3] proposed a 2D texture-based method which uses compressed texture atlas to reduce interpolation costs. Nogera et al. [4] proposed a webGL application to visualize very large 3D volumes by using multi-texturing to encode volumetric models on a set of RGBA 2D textures. A recent application developed by Balsa et al. [5] allows to interact with volume models using mobile devices hardware. Their scheme is not compressing the volume data.

In other schemes, the transmitted data is a compressed image, the transfer function is applied at the beginning of the pipeline, followed by a 2D rendering on a texture, all done on the server side. A compressed image is sent to the client where decompression and image rendering takes place. This scheme is frequently named "Thin Clients" [6]. The idea in multiresolution model schemes [7] is to render only a region of interest at high resolution and to use progressively low resolution when moving away from that region. Both bricking and multiresolution approaches [8] need a high memory capacity on the CPU for storing the original volume dataset. Moreover, bricking requires a high amount of texture transfers as each brick is sent once per frame; multiresolution techniques have been built for CPU purposes and its translation to GPUs is not straightforward due to the required number of texture accesses.

Preprocessing of data is also a useful technique, as it ensures the reduction of the information, combined with different techniques for quantization, encoding and multiresolution representation [8].

Efficient schemes require optimized algorithms to reduce and send data through the network. The algorithms must achieve the maximum compression possible while allowing an easy decompression in the client side, where sometimes hardware and memory constraints decrease performance [8].

Wavelet transforms offer considerable compression ratios in homogeneous regions of an image while conserving the detail in non-uniform ones. The idea of using 3D wavelets for volume compression was introduced by Muraki [9]. Ihm and Park [10] proposed an effective 3D 16^3-block-based compression/decompression wavelet scheme for improving the access to random data values without decompressing the whole dataset. Guthe et al. [11] proposed a novel algorithm that handles a hierarchical wavelet representation where decompression takes place in GPU.

Some techniques advocate the use of hybrid region-based volume rendering, by applying different shading algorithms inside the volume model [12], or by implementing multiresolution region-based schemes [1]. Luo *et al.* [13] developed a technique for focusing on a user-driven ROI while preserving context information. The approach uses a distance function to define the region of interest. This function controls voxel opacity, exploits silhouette enhancement and non-photorealistic shading.

In this paper, we propose a hybrid framework that exploits the use of standard transfer functions as an alternative to compress volume dataset. Our scheme is a transfer function-aware scheme for client/server techniques. It combines Wavelet-preprocessed volume data to reduce information outside the ROI, and highlighted segmented data in regions of interest (ROI), (Gradient Octree shceme). From the best of our knowledge this possibility has not been considered by any of the described approaches in this previous work.

3 OVERVIEW OF THE APPROACH

Let us assume that we are interested in inspecting a volume data model V which is too large in terms of network transmission and/or client storage facilities. Wavelet compression algorithms like the ones presented in [1, 10, 11] are able to support block-based regions of interest (ROIs). Other approaches like Gradient Octrees [2] can be rendered with advanced illumination models and at a higher visual quality level. Gradient Octrees are specific data structures for multiresolution volume datasets. Gradient Octrees $G(V)$ include an specific data structure S and a compact data array D. The octree structure S can be sent to the client devices in a lossless way with only one bit per node, whereas data is compacted to 3 Bytes per octree node, including material information and volume gradients. Both approaches, however, have advantages and drawbacks:

(I) TF-aware wavelet compression schemes succeed in sending to the clients a very limited amount of information in the areas outside the region of interest (ROI). High quality volume information in the ROI is also compact [1], because the 3D texture is smaller and restricted to the blocks in the ROI area. Ray-casting visualization in the client can use compact 3D textures which are suitable for many client devices. The main drawback of this approach, however, is twofold. First, changing the transfer function requires sending a new version of the compress volume model to the client, and second, it is not well suited for illumination computations that would require too many texture accesses.

(II) Approaches like Gradient Octrees overcome these limitations by supporting multiple transfer functions and materials and by precomputing gradients on the server. They support advanced illumination models, thus achieving a higher visual quality level. However, they are not direct candidates for ROI-based visualization paradigms, as their low level volume representations present a flat-face appearance with poor gradients. These representations at coarse tree levels are

well suited for progressive transmission but they perform worse than similar-quality low-level wavelet reconstructions.

Our hybrid scheme inherits the best of both approaches. In this case, apart from the volume model V, the user must supply a set of transfer functions $\{TF_k\}$ and selects one of them as a canonical transfer function. The server starts by computing a Gradient Octree $G(V)$ from V and for the set $\{TF_k\}$, also computing the quantified representation $W(V)$ of the wavelet transform of V with the canonical transfer function, as described in Section 4. $G(V)$ encodes materials and gradients only in the subset of voxels of V which are relevant to same of the transfer functions in $\{TF_k\}$.

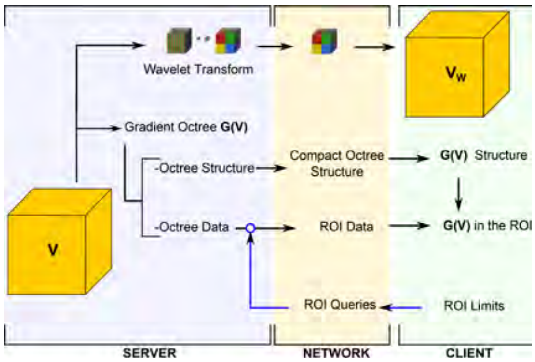


Figure 1: Overview of the proposed scheme, showing the preprocess on the server, the data transfer through the network and the data structures in the client device

Users at the client side can interactively define regions of interest, ROIs. Information over the network can be classified into *static information* (being send only once per volume model) and *dynamic information*. Dynamic information must be re-sent whenever the ROI is redefined by the user. Static information is compact, including $W(V)$ and a set of arrays defining the tree structure of $G(V)$. In cases where the size of the volume model V is too large and the volume data at the deepest level of $G(V)$ does not fit into the client's CPU memory, the portion of this data belonging to the ROI is generated from the octree data on demand, as *dynamic information*, whenever the user asks for a different ROI.

In the client side, a low-resolution volume model V_W is reconstructed by de-quantizing and computing a few inverse wavelet steps in each block. Let us note as V_R the subvolume corresponding to the ROI. A two-level ray-casting rendering algorithm in the client GPU (Section 5) succeeds at showing a high-quality Gradient Octree rendering in V_R together with a visualization of V_W in the parts of the volume outside the ROI, also supporting a number of interaction facilities.

The corresponding compression and decompression algorithms are detailed in Section 4.

4 COMPRESSION AND DECOMPRESSION ALGORITHMS

We start by computing the wavelet transform $W(V)$ of the volume model V and its gradient octree $G(V)$ on the server, Figure 1. We use a localized, block-based transform with a previous smoothing step to achieve local behaviour and a better compression rate. We assume standard piecewise linear transfer functions [**?**]. By considering these transfer functions, we virtually segment the volume V in as many regions as linear segments defined by the $\{TF_k\}$ functions. Voxels with a density d such that the opacity of $\{TF_k\}$ is zero for all k, belong to null regions and are simply represented by a null code. Our implementation uses a block size of 16 together with a 4-steps Haar transform, being rather efficient in compression while supporting block-aware interaction paradigms in the client. As already mentioned, the wavelet information that is sent over the network to the client is a low resolution volume model V_W, obtained by computing a few w_l inverse wavelet steps in each block. Observe that in the usual case of $w_l = 2$, the size of the information in V_W will always be lower than 1/64 of the size of the initial model V. The low-resolution model for $w_l = 2$ is compressed more than a 98.5%. In what follows, we will use the term *compression rate* to name the relative size of the compressed model, which in this case is 1.5%.

The gradient octree $G(V)$ information includes the octree structure and the octree data. Creating $G(V)$ involves three compression steps. The first is transfer-function aware and uses V and the set of $\{TF_k\}$ to compute an Edge Volume model $VE(TF)$ which only encodes voxels that are relevant to the transfer functions $\{TF_k\}$. Non-relevant voxels in $VE(TF)$ are assigned a Nil value. On a second step, we compress gradient information to a total of three bytes per Grey tree node (including material information) in a set of data arrays, one per octree level, [2]. We use a GPU-oriented encoding of the proposed hierarchical data structure with explicit volume gradient information in octree nodes, to avoid gradient computations during GPU ray-casting. The final Gradient Octrees representation, shown in Figure 2, consists on a small volume model V_{32} with pointers and two sets of per-level arrays, O_l and D_l. For the sake of clarity, octree levels in Figure 2 and in what follows will be identified by their resolution. The example shown in Figure 2 corresponds to the complete octree representation of a volume V of resolution $r = 512$, with gradient and materials stored in the data array D_{512}. Data arrays of coarser octree levels (D_{256}, D_{128}, D_{64} and D_{32}) store gradient and materials data of Grey octree nodes at these levels. In short, the octree structure S of $G(V)$ consists of the pointers volume V_{32} and the set of per-level arrays O_l. The octree data D of $G(V)$ includes the set of per-level arrays D_l.

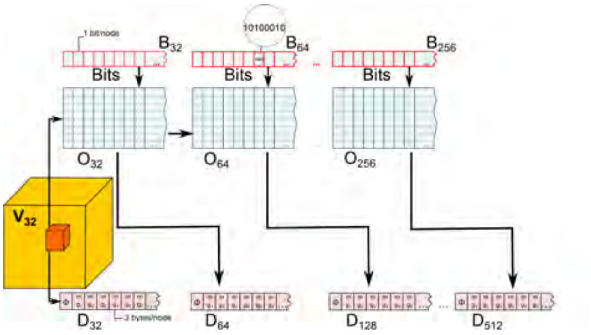


Figure 2: Encoding Gradient Octrees

For transmission purposes, it is possible to compact the Gradient Octree structure (not the volume data) in a lossless way. Instead of sending the arrays O_l, we send a set of arrays B_l with one byte per parent node as shown in red in Figure 2. Note that this is equivalent to store and transmit only one bit per node: the components in B_l simply represent the type of each child in one bit (0 if Nil, 1 if Grey). Compressed arrays B_l are computed on the server and sent to the client. For every received level, the client is able to generate a B_l-driven, increasing sequence of indexes to create a local copy of the array of indexes to child nodes O_l (for a detailed discussion, see [2]). Note that O_l indexes point simultaneously to D_{2l} and to O_{2l}. The volume V_{32} is sent to the GPU as a 3D texture, whereas arrays O_l and D_l are encoded as 2D and 1D textures. In short, we succeed in sending the tree structure S in a lossless way and with only one bit per node, through a sequence of compact arrays B_l. Moreover, compressing gradients and materials in three bytes is efficient, supports GPU decompression and suffers from a very limited loss in visual quality.

Although ROI-dependent localizations of the octree structure S could be defined [14], we have observed that the corresponding compression improvements (mainly in the information over the network) are negligible. In our present implementation we have therefore considered a hybrid model consisting of the low-resolution volume V_W, the gradient octree structure S and ROI-dependent octree data D. This hybrid model information is sent from the server to the client (or clients) in two parts: (I) The static information is sent only once, at the beginning of the interaction session. It consists on the low-resolution volume V_W, the $32 \times 32 \times 32$ pointers volume V_{32} of the computed Gradient Octree, the set of arrays B_l which encode the S octree structure and the materials look-up table of the Gradient Octree, Figure 1. The size of this last table is very small and we will not consider it in our compression computations. (II) The dynamic information is sent on demand whenever the client changes the ROI. The client sends a query with the new ROI limits (bounding box) and the server generates and sends a subset D_R of the data arrays D of the Gradient Octree, as we know in advance that only voxels in the ROI will be retrieved and rendered, Figure 1.

In our present implementation we assume that users are only interested in gradient octree data at the deepest octree level r, as lower resolutions are already shown outside the ROI. This makes the whole process easier, as we can just send a compact D_r array containing only those voxels with a non-Nil gradient value in the deepest octree level. This results in a very compact data transmission. We compute and keep a temporal, ROI-dependent version of the pointers volume V_{32} which we name VR_{32}. VR_{32} has Nil pointers outside the ROI and sequential pointers for the Grey nodes inside the ROI. The textures VR_{32} and O_k are now ROI-dependent, and must be recomputed in the client from the $G(V)$ Structure (see Figure1) whenever the ROI is changed during the interaction, with a very efficient algorithm which only involves array traversal and counting.

5 RENDERING AND INTERACTION IN THE CLIENT DEVICES

Reconstruction of any of the blocks within the non-ROI partion of the volume can be performed at one, two, three or four wavelet levels. The four-level reconstruction of a block generates a full piece of $16 \times 16 \times 16$ voxels that represent the corresponding part of the volume. Reconstructions of the same block at three, two or one levels generate pieces of $8 \times 8 \times 8$, $4 \times 4 \times 4$ or $2 \times 2 \times 2$ voxels, representing the same part of the volume at lower resolutions.

A usual interactive session starts by inspecting the whole volume model at a low resolution. In this case, all blocks are usually reconstructed at one or two levels, the corresponding 3D Texture is sent to the client GPU and ray-casting rendered. Observe that the size of this 3D texture, in the case of two reconstruction levels, is 1/64 of the size of the original volume model V.

Alternatively, the user can decide to inspect the whole volume model at a low resolution (two levels of reconstruction, for instance) with the ROI showing predefined structures at maximum level of detail by ray-casting the Gradient Octree. To achieve this last interactive visualization, two structures, one for the non-ROI volume (3D texture) and the other for the ROI volume (Gradient Octree), are sent to the client GPU where an adaptive ray-casting algorithm is performed, as detailed below. Since the whole model is available at the client side, rotation and zooming operations can be autonomously performed in the client without any further transmission from the server. If a region of the model needs to be detailed, the Gradient Octree can be displayed on demand.

We use a standard ray casting algorithm in the client GPU, the main difference with the classical algorithms

being that rays traverse a low resolution 3D texture representing the whole volume. In the point samples along the ray that do not belong to the ROI, the ray-casting uses density values from the low-resolution model V_W. Samples in the ROI retrieve densities from a virtual volume with the same resolution R as V_R, instead of traversing V_R itself. In Figure 3, an example with resolution $R = 512$ is shown. Ray-casting proceeds as usual by advancing along rays r from the observer with a uniform sampling of the volume along r. Then, for each sample s of r addressing a virtual voxel (i, j, k) in the ROI, its volume information is found in D_R.

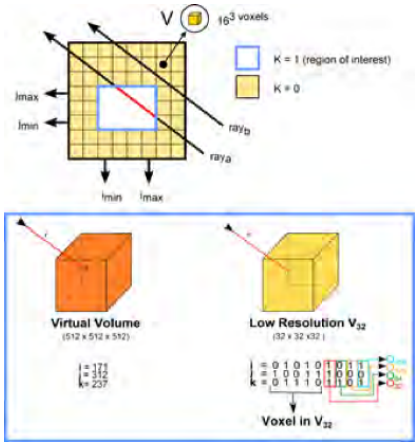


Figure 3: The block structure of the model, a region of interest (in white) and the octree-based ray-casting.

Ray casting within the ROI is based on the octree addressing properties. The octree search of any virtual voxel (i, j, k) is directly driven by the base-2 representation of i, j and k, as shown in Figure 3. In this case, their first 5 binary digits point to the corresponding voxel in the low-res texture V_{32}. The index i_{32} found in this V_{32} voxel element points to the low resolution data in D_{32} (which we don't use if a higher resolution is required) and also to the array of its eight child indexes in O_{32}. A well-known property of binary octree subdivision ensures that next "three bit columns" in the binary representation of i, j, k are in fact child indexes s_l. Son indexes point to deeper octree levels and are able to drive the octree traversal to the right element in D_R containing data in the virtual volume voxel. Subtree traversal from the low-res voxel in V_{32} to the virtual voxel data is based on the recursion equation,

$$i_{2l} = O_l[i_l][s_l] \tag{1}$$

for l=32, 64, 128, .. $R/2$

The final index i_R points to the high-res data in D_R, but tree traversal can stop earlier if the virtual voxel is void and any index i_l in the chain is found to be zero.

Observe that only virtual voxels in V_R in the ROI will be addressed. This means that the client must only store a restriction of $G(V)$ in V_R. In our present implementation we initially send the whole octree structure of $G(V)$ to the client, but high-resolution data D_R (restricted to V_R) is only sent on demand when the user changes the ROI

Let's assume that ray r is crossing voxel $i = 171$, $j = 312$, $k = 237$ in the virtual volume of the ROI, Figure 2. In this case, the octree search starts in the voxel $(10, 19, 14)$ of V_{32} and is then driven by four child indexes: $s_{32} = 7$, $s_{64} = 1$, $s_{128} = 4$ and $s_{256} = 5$ which recursively generate the indexes i_{64}, i_{128}, i_{256} and i_{512}. Reaching the deepest level information in a Gradient Octree of resolution $R = 512$ involves a maximum of six texture queries, to V_{32}, O_{32}, O_{64}, O_{128}, O_{256} and finally to D_{512}.

After retrieving high-res data in D_{512}, materials and the gradient vector are decompressed on the fly in the GPU. Obviously, everything also works when lower resolution virtual volumes are considered.

6 RESULTS AND DISCUSSION

To perform a complete comparative study , we selected the following accessible frameworks:

VrMed Viewer, an integration of libraries and functionalities, designed to achieve interactive visualization in PCs and Virtual Reality Systems, using a GPU-based Ray-casting algorithm [15].

Volume Viewer, an Android based application [5], implemented to run on mobile devices. Allows interactive visualization of models with a transfer function editor with easy handling. In this case, the whole volume model is sent to the client device.

VrMed-Thin Client The approach is based on [16]. It achieves remote visualization of volume models with basic user interaction tools in mobile devices. A server running VRMed Viewer on Linux operative system, renders images which are sent to the client through the wireless. Clients generate control commands as OpenGL parameters which are sent to the server using a TPC/IP socket.

Tables 1 and 2 show a comparison of [2], [1], our technique (**Hybrid Approach**), and the previously described schemes, using two models: The skull model with a $256 \times 256 \times 112$ resolution and the thorax model with a 512^3 resolution. Density values are in the rang [0...255], hence each voxel is codified using only 1 byte. Table rows show for each scheme whether multiresolution and progressive transmission is allowed and the compression rate achieved for each case. Both tables also show the size of the transmitted data trough the network and the client requirements to perform volume rendering followed by an estimation of the average frame rates in two cases: rendering in the PC server and rendering in the client (mobile device).

Figure 4 shows some snapshots of the interaction with the hybrid skull model. In all cases, two wavelet reconstruction steps ($wl = 2$) have been used without lighting computation in the low resolution area: In (a), (b) and (d) the Wavelet have been computed by using a skin tranfer function, whereas the image in (c) represents a bone transfer function both in the low resolution area and in the Region of Interest.

Some snapshots of the interaction with the hybrid thorax model are shown in Figure 5. In all cases two wavelet reconstruction steps ($wl = 2$) have also been applied without lighting computation, in the low resolution area: ROI showing ribs and lungs (a), internal gases and lungs (b) and skin, ribs and lungs (d). The snapshot in (e) shows the alveoli in the ROI, magnified in (f) by interacting with zoom and a section plane. In these cases wavelets are precomputed after applying to the model a TF covering all structures in the low resolution area. The snapshot in (c), shows a TF for bones visualization in both, the low resolution area, and the ROI. Image in (d) shows a zoom-in of (c) for showing up the quality of the hybrid model. The server application runs on PC with 6 GB of RAM, Intel Core 2 Duo at 3.16 GHz and a client with 4 GB of RAM, Intel Core 2 Duo at 3.06 GHz and Nvidia GeForce GTX z80. Client tests were performed on the HTC One smartphone whith a screen resolution of 1080 x 1920 pixels 2 GB RAM and an Adreno 320 Graphics processor.

A ROI-based visualization has been considered in the Wavelets-based scheme and in the Hybrid approach, while in the rest of columns, the whole volume V has been rendered at a uniform resolution. This is valid in both cases (tables 1 and 2). Zoom has been adjusted in a way that the total amount of rendered ROI pixels in the application viewport is a 25% of the total of viewport pixels. The amount of ROI pixels in the viewport is relevant, as it measures the total amount of required high-quality casted rays during ray-casting rendering.

Compression rates correspond to the amount of data sent over the network, and relate this amount to the total memory requirements of the volume models, which are 7.4 MBytes in Table 1 and 128 MBytes in the case shown in Table 2. In contrast to the previous schemes, our techniques allow multi-resolution rendering with progressive transmission of volume data. For the Wavelet based approach, the presented figures on the amount of data over the network represent the necessary information to reconstruct four levels of wavelet compression, $w_l = 4$. The compression rate in this case is between 21% and 32%.

In case of the Gradient Octree approach, data includes the octree structure plus material and gradient information at its deepest, maximum resolution level. In the Hybrid scheme, the *information over the network* represent both the necessary data to reconstruct two levels of wavelet compression for the low resolution model and the nodes representing the Gradient Octree leaves in the selected region of interest (ROI). This approach also requires a client GPU being able to manage 3D textures. The compression rate in this case is between 20% and 22%, with an average frame rate in the mobile device between 7 and 16 fps.

The proposal in this paper **Hybrid approach** results in a compression rate which is between 4% and 18%, with an average frame rate in the mobile device between 8 and 16 fps when $w_l = 2$. It also requires a client GPU being able to manage 3D textures.

The VrMed viewer is presented for comparison purposes. Some of the parameters in the tables do not apply to this case, as VrMed is a stand-alone application without network transmission. The average frame rates, 48 and 20 fps, are obviously higher than those in the previous cases but these figures show that our proposed approaches are performing within reasonable efficiency limits.

The Thin Client based approach sends a maximum of 0.18 MB of data through the network per frame during an interactive session with a single client (of course, the total amount of transmitted data depends on the number of interaction frames). This is due to the fact that the technique requires the transmission of rendered images from the server when the user interacts with the model in the client side. This fact makes this scheme network dependent, with framerates which decrease in network congestion cases. We have observed that our thin-client implementation becomes useless when the number of clients is above 8. On the other side, this scheme does not require sophisticated client GPUs, as clients must only decompress and show pre-rendered images. This can be an advantage for basic client devices, but result in an under-utilization of client GPUs in the case of most present devices. The asterisks in the *Thin client* column in tables 1 and 2 mean that data sizes are per frame sizes. The compression rates obviously depend on the number of transmitted frames.

The Hybrid approach is specially well suited in the case of large models. The comparison between tables 1 and 2 show that this is a scalable scheme, with compression rates that decrease when the size of the volume model increases. The corresponding frame rates are larger than in the case of Gradient Octrees, being of the same order of magnitude than Thin Clients.

Comparing Thin clients to Wavelets, Gradient Octrees and the Hybrid approach, we can define the break-even interaction period as the number of frames required to have an equivalent amount of information sent over the network. Break-evens are computed as the ratio between the size of the compressed model as sent over the network in our approaches and the size of a single Thin Client frame image. In the case of the skull model

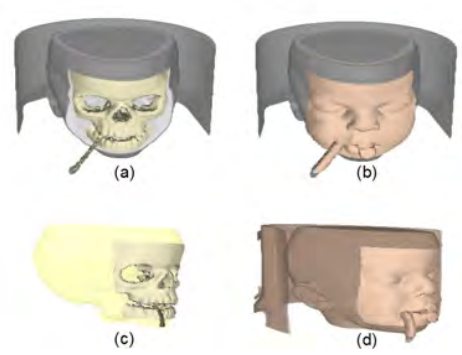


Figure 4: Hybrid Visualization. Interaction with the hybrid skull model. ROI size: $(128 \times 64 \times 64)$.

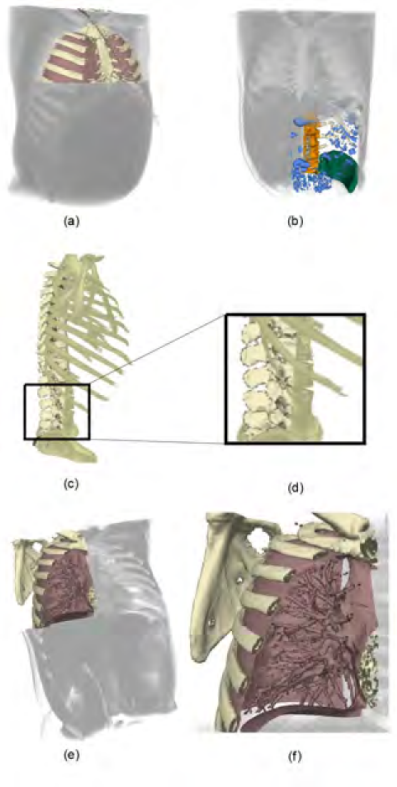


Figure 5: Hybrid Visualization. Interaction with the hybrid thorax model. ROI size: $416 \times 224 \times 224$ (a), $256 \times 160 \times 256$ (b), $96 \times 128 \times 480$ (c), and $256 \times 512 \times 256$ (d), (e) and (f).

in Table 1, this break-even is 11 frames for Wavelets, 21 frames for Gradient Octrees and 11 frames for the Hybrid approach.

In the case of the thorax model in Table 2, the break-even is 51 frames for Wavelets, 79 frames for Gradient Octrees and 17 frames for the Hybrid approach. By considering the number of frames per second in each case, we can conclude that the information we are sending is equivalent to the total information sent by the Thin Client approach during an interaction period in between 1 and 10 seconds. In the case of the Hybrid approach, break-evens are 11 frames and 17 frames, meaning this Hybrid scheme outperform Thin Clients in interaction cases longer than around 20 frames.

Thin Clients can also be compared with the presented approach in terms of frame rates. Frame rates depend on the network bandwidth, the present approach being better than Thin Client approaches in geographic regions with a limited bandwidth. In fact, the presented proposal can be specially useful in world regions with limited network infrastructures but requiring fast access to 3D medical data, like non-urban areas.

The Volume Viewer approach as presented in the last column of both tables does not require sophisticated client GPUs, as clients are rendering stacks of 2D textures. Frame rates in the client are reasonable. The main drawback in this case, however, is the amount of information being sent over the network, which makes it unusable in the case of large volume models.

The proposed hybrid scheme allows interactive inspection by rotating and zooming volume models. Users are able to select ROI portions of the visualized model, as well as choosing a transfer function from the set of transfer functions ($\{TF_k\}$) inside the the selected ROI. Interface options include also, planar section selection and offsets structures visualization in front of the selected section plane. Users can also choose the resolution of the low resolution region, by reconstructing models, by using one, two or three wavelet reconstruction steps.

	Wavelets Sheme	Gradient Octrees	Hybrid Approach	VrMed Viewer	VrMed Thin Client	Volume Viewer
Multi-resolution				no	no	no
Progressive transmission				no	no	no
Compression rate (%)	32	22	18	-		100
Data over the network(MB)	1.91	3.8	2.04	-	0.18*	7.3
Client requirements	3D Tex	3D Tex	3D Tex	-	Image	Stack 2D Tex
Frame rate (PC version)	52	24.12	46.53	48.24	48.24	-
Frame rate (mobile)	24.2	-	16.32	-	20.23	17.0

Table 1: Comparison between the approach presented in this paper (blue column) and some previous schemes for remote visualization of volume models. Study of the Skull model with a resolution of $256 \times 256 \times 112$ and 7.3 MB of size.

	Wavelets Sheme	Gradient Octrees	Hybrid Approach	VrMed Viewer	VrMed Thin Client	Volume Viewer
Multi-resolution				no	no	no
Progressive transmission				no	no	no
Compression rate (%)	21	20	4.2	-		100
Data over the network(MB)	16.4	27	5.3	-	0.32*	128
Client requirements	3D Text	3D Tex	3D Tex	-	Image 2D Tex	Stack
Frame rate (PC version)	14	10.31	18.03	20.34	20.34	-
Frame rate (mobile)	13.20	-	8.07	-	20.23	-

Table 2: Comparison between the approach presented in this paper (blue column) and some previous schemes for remote visualization of volume models. Study of the Thorax model with a resolution of 512^3 and 128 MB of size.

7 CONCLUSIONS & FUTURE WORK

We have proposed a Hybrid approach that inherits the advantages of the algorithms presented in [1] and [2] while keeping a good performance in terms of bandwidth requirements and storage needs in client devices. Information over the network consists on static information (being only set once) and dynamic information. Dynamic information must be re-sent whenever the ROI is redefined by the user. The complexity (memory and data transmission requirements) of the static and dynamic information has been discussed. The main conclusion is that the hybrid scheme is flexible enough to represent several materials and volume structures in the ROI area at a very limited static and dynamic information transmission cost.

The Hybrid approach has been proved to be specially well suited in the case of large models. The presented experimental tables show that the Hybrid approach is a scalable scheme, with compression rates that decrease when the size of the volume model increases. Corresponding frame rates are larger than in the case of Gradient Octrees, being of the same order of magnitude than Thin Clients. Our compression results are better than similar client server schemes for volume rendering, and compare favourably to Marching Cubes based approaches. While these last schemes must send an average of three triangles per voxel in segmented volumes, the presented approach only sends three bytes/voxel. We consider that our approach may enrich the user experience during the inspection of volume medical models in these low performance devices.

ACKNOWLEDGMENT

This work has been partially supported by the Project TIN2013-47137-C2-1-P of the Spanish Ministerio de Economía y Competitividad.

8 REFERENCES

[1] Lázaro Campoalegre, Pere Brunet, and Isabel Navazo. Interactive visualization of medical volume models in mobile devices. *Personal Ubiquitous Comput.*, 17(7):1503–1514, 2013.

[2] Lázaro Campoalegre, Pere Brunet, and Isabel Navazo. Gradient octrees: A new scheme for remote interactive exploration of volume models. *Proc. of the CAD/Graphics*, 2013.

[3] Manuel Moser and Daniel Weiskopf. Interactive volume rendering on mobile devices. In *Vision, Modeling, and Visualization VMV*, volume 8, pages 217–226, 2008.

[4] Jose Maria Noguera and Juan Roberto Jiménez. Visualization of very large 3d volumes on mobile devices and webgl. In *WSCG international conference on computer graphics, visualization and computer vision*, 2012.

[5] Marcos Balsa and Pere Vázquez. Practical volume rendering in mobile devices. In *Advances in Visual Computing*, pages 708–718. Springer, 2012.

[6] Klaus Engel and Thomas Ertl. Texture-based volume visualization for multiple users on the world wide web. In *Virtual Environments*, pages 115–124. Springer, 1999.

[7] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization*, pages 355–361, 1999.

[8] Enrico Gobbetti, José Antonio Iglesias Guitián, and Fabio Marton. Covra: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. In *Computer Graphics Forum*, volume 31, pages 1315–1324, 2012.

[9] Shigeru Muraki. Volume data and wavelet transforms. *Computer Graphics and Applications, IEEE*, 13(4):50–56, 1993.

[10] Insung Ihm and Sanghun Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, pages 3–15, 1999.

[11] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer. Interactive rendering of large volume data sets. In *Visualization, 2002. VIS 2002. IEEE*, pages 53–60. IEEE, 2002.

[12] Jianlong Zhou, Manfred Hinz, and Klaus D Tönnies. Hybrid focal region-based volume rendering of medical data. In *Bildverarbeitung für die Medizin 2002*, pages 113–116. Springer, 2002.

[13] Yanlin Luo, José Antonio Iglesias Guitián, Enrico Gobbetti, and Fabio Marton. Context preserving focal probes for exploration of volumetric medical datasets. In *Proceedings of the International Conference on Modelling the Physiological Human*, pages 187–198, 2009.

[14] Lazaro Campoalegre. Contributions to the interactive visualization of medical volume models in mobile devices. In *PhD Thesis, UPC*, 2014.

[15] Eva Monclús, José Díaz, Isabel Navazo, and Pere-Pau Vázquez. The virtual magic lantern: An interaction metaphor for enhanced medical data inspection. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 119–122, 2009.

[16] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 109–118, 2000.

Joint Bilateral Mesh Denoising using Color Information and Local Anti-Shrinking

Oliver Wasenmüller
oliver.wasenmueller@dfki.de

Gabriele Bleser
gabriele.bleser@dfki.de

Didier Stricker
didier.stricker@dfki.de

DFKI GmbH - German Research Center for Artificial Intelligence
Augmented Vison Department
Trippstadter Str. 122
67663 Kaiserslautern, Germany

ABSTRACT

Recent 3D reconstruction algorithms are able to generate colored meshes with high resolution details of given objects. However, due to several reasons the reconstructions still contain some noise. In this paper we propose the new Joint Bilateral Mesh Denoising (JBMD), which is an anisotropic filter for highly precise and smooth mesh denoising. Compared to state of the art algorithms it uses color information as an additional constraint for denoising; following the observation that geometry and color changes often coincide. We face the well-known mesh shrinking problem by a new local anti-shrinking, leading to precise edge preservation. In addition we use an increasing smoothing sensitivity for higher numbers of iterations. We show in our evaluation with three different categories of testdata that our contributions lead to high precision results, which outperform competing algorithms. Furthermore, our JBMD algorithm converges on a minimal error level for higher numbers of iterations.

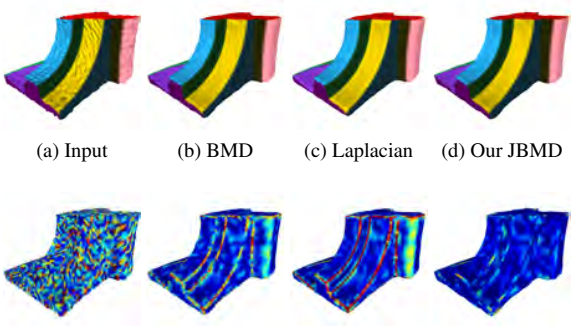
Keywords

Mesh Denoising, Smoothing, Fairing, Joint Bilateral Filter, Local Anti-Shrinking, Color Information.

1 INTRODUCTION

Many applications, such as urban planning, industrial measurement or human anthropometry, require reconstructed 3D models of the respective objects with very high precision. The traditional approach is to acquire these models by laser scanners, since they promise high quality results. However, they are expensive, impractical to use and contain still some noise. Meanwhile 3D reconstruction algorithms, such as e.g. [Agi, FP10, NIH+11, SSC14], are able to generate colored mesh models from devices like standard color cameras and\or depth cameras, which are widely spread, cheap and easy to use. These reconstructions contain color information together with high-resolution details, but also suffer from noise in their 3D geometry. To get rid of this noise, several (mostly iterative) methods for mesh denoising were proposed in the literature. Sometimes they are also referred to as smoothing, filtering or fairing methods. They use directly the 3D geometry or derived measures, like distances or normals of the mesh, to estimate new vertex positions. However, none of them explicitly uses the color information provided by e.g. one of the above mentioned algorithms. Thus, we present in this paper a new anisotropic method called Joint Bilateral Mesh Denoising (JBMD), which uses - besides geometric information - the color information as an additional constraint for edge preserving denoising.

Another well-known problem of mesh denoising are shrinking effects. They occur mostly in curves regions of the mesh and are caused by homogeneous shifts of vertices in a neighborhood into one major direction. Current approaches try to compensate that ef-



(a) Input (b) BMD (c) Laplacian (d) Our JBMD

Figure 1: Comparison of different mesh denoising algorithms for the *fandisk* mesh. Top row: meshes. Bottom row: color-coded error distribution.